(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開平11－161504
(43)公開日　平成11年(1999)６月18日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｄ

審査請求　有　　請求項の数6　ＯＬ　（全９頁）

(21)出願番号　特願平10－275834
(22)出願日　平成10年(1998)９月29日
(31)優先権主張番号　９７　１２１４６
(32)優先日　1997年９月30日
(33)優先権主張国　フランス（ＦＲ）

(71)出願人　390035633
ブル・エス・アー
フランス国、エフ－78434・ルーブシエンヌ、ルート・ドウ・ベルサイユ、68
(72)発明者　ダニエル・リユシアン・デユラン
フランス国、78180・モンテイニー・ル・ブルトヌー、リユ・マリア・カラス・38
(72)発明者　ジエラール・シトボン
フランス国、94400・ビトリー、リユ・ガニエ・12
(72)発明者　フランソワ・ユルバン
フランス国、75002・パリ、リユ・ドウ・パレストロ・33
(74)代理人　弁理士　川口　義雄　（外２名）

(54)【発明の名称】　情報処理システムにおけるジョブの実行を処理する装置および方法

(57)【要約】
【課題】　オープン情報処理システム上でのジョブの実行を、資源の量に応じて処理する方法および装置を提供する。
【解決手段】　本発明方法は、仮想メモリ、実メモリ、一時的なファイルスペース、最後の時間間隔の間のＣＰＵの利用時間として使用可能な資源を決定し、予め他の要求に割り当てられている未使用の資源の量を算出し、要求が提示されたジョブの実行に必要な資源の量と、予め他の要求に割り当てられた資源の量が差し引かれた利用可能な現在の資源の量とを比較し、この比較の結果に応じて、要求されたジョブのスタート、延期またはスタートの拒否を決定するステップを含む。

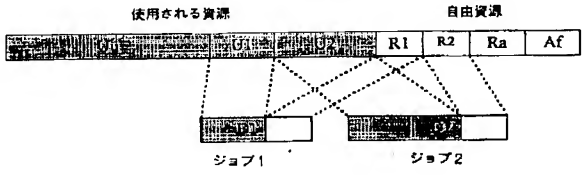


FIG. 2

1

【特許請求の範囲】
【請求項１】　オープン情報処理システム上でジョブの実行を資源に応じて処理する方法において、
仮想メモリ、実メモリ、一時的なファイルスペース、最後の時間間隔の間のＣＰＵ使用時間として利用可能な資源を決定し、
予め要求に割り当てられている未使用の資源の量を算出し、
要求が提示されたジョブの実行に必要な資源の量と、予め他の要求に割り当てられた資源の量が差し引かれた利用可能な現在の資源の量とを比較し、この比較の結果に応じて、要求されたジョブのスタート、延期またはスタートの拒否を決定する、ステップを含むことを特徴とする方法。
【請求項２】　ジョブの実行に必要な資源の量を特定のコマンドによって規定することからなるステップを含むことを特徴とする請求項１に記載の方法。
【請求項３】　利用可能なシステム資源が充分にあるか否かを決定する機構を、作動させるあるいは作動解除させることからなるステップを含むことを特徴とする請求項１または２に記載の方法。
【請求項４】　オープン情報処理システム上でジョブの実行を処理する装置において、
仮想メモリ、実メモリ、一時的なファイルスペース、最後の時間間隔の間のＣＰＵ使用時間として利用可能な資源を決定する手段と、
予め他の要求に割り当てられている未使用の資源の量を算出する手段と、
要求が提示されたジョブの実行に必要な資源の量と、予め他の要求に割り当てられた資源の量が差し引かれた利用可能な現在の資源の量とを比較する手段と、
比較手段によって供給される結果に応じて、要求されるジョブのスタート、延期またはスタートの拒否を制御する手段とを含むことを特徴とする装置。
【請求項５】　ジョブの実行に必要な資源の量を特定のコマンドによって規定する手段を含むことを特徴とする請求項４に記載の装置。
【請求項６】　利用可能なシステム資源が充分にあるか否かを決定する機構を、作動させるあるいは作動解除させる手段を含むことを特徴とする請求項４または５に記載の装置。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、情報処理システムにおけるジョブの実行を処理する装置および方法に関する。
【０００２】
【従来の技術】本発明は、工業環境における情報処理オペレーション分野、特に「ユニックス」または「ウィンドウズＮＴ」タイプのソフトウェア上で動作する「オープン」型の情報処理システムに適用される。このタイプのソフトウェア上で動作するサーバは、いわゆる「オープン」型サーバであり、今日、特に重視されているものである。

2

【０００３】これらのサーバの不都合は、目的が工業的目的ではない学術的アプリケーションのために開発されたことにある。このように、「ユニックス」タイプのＯＳでは、ジョブあるいはより正確には、ジョブを構成する各プロセスには、システムまたは、これらのジョブを入力するユーザによって初期優先度が割り当てられる。システムが一つのジョブにより高い優先度を与えるようにするためには、ユーザは、初期の優先度をジョブに明示的に割り当てるか、管理コマンドによってその優先度を手動で変えなければならない。従って、ジョブに割り当てられる優先度レベルの間には、いかなる調整も相互関係もない。
【０００４】同様に、標準的「ユニックス」システムにおいては、プロセスは、非常に負荷が大きいシステムでも制限なしで始動することができ、実行プロセスが現在使用している資源の量には注意を払う必要はなく、また新しいプロセスが必要とする資源の量にも全く注意を払う必要がない。これによって次のような好ましくない挙動が起きる。すなわち重要なプロセスであろうと重要でないプロセスであろうと一連のプロセスが実行され、ユーザが所望の時間に終了しなければならない重要なプロセスが使える資源は、この所望の時間に終了するには少なすぎ、同時に、重要でないプロセスが多すぎる資源を使用する。
【０００５】容認しがたいもう一つの挙動は、利用可能な一時的ファイルが充分でなく、その結果としてファイルへの書き込みに失敗することである。アプリケーションは、こうしたエラー条件を制御するときもあれば、制御しないときもあるので、不完全なファイルがジョブのあるステップで発生し、その後のステップで処理されることから、誤った挙動を起こすことがある。別の有害な挙動は、ページングスペースが充分にないことであり、システムは、最も最近のプロセスを、その重要性を考慮せずに消去する決定を行い得る。これは、オペレーション環境では受け入れられない。
【０００６】
【発明が解決しようとする課題】このような理由から、本発明の第一の目的は、これらの欠点を解消可能なジョブの実行を考慮する方法を提案することにある。
【０００７】
【課題を解決するための手段】この目的は、（たとえば「ユニックス」タイプの）オープン情報処理システム上でジョブの実行を資源に応じて処理する方法によって達成され、この方法は、仮想メモリ、実メモリ、一時的なファイルスペース、最後の時間間隔の間のＣＰＵの使用時間として利用可能な資源を決定し、予め他の要求に割

り当てられている未使用の資源の量を算出し、要求が提示されたジョブの実行に必要な資源の量と、予め他の要求に割り当てられた資源の量が差し引かれた利用可能な現在の資源の量とを比較し、この比較の結果に応じて、要求されたジョブのスタート、延期またはスタートの拒否を決定するステップを含むことを特徴とする。

【０００８】他の特徴によれば、この方法は、ジョブの実行に必要な資源の量を特定のコマンドによって規定することからなるステップを含む。

【０００９】別の特徴によれば、この方法は、利用可能なシステム資源が充分にあるか否かを決定する機構を、作動させるあるいは作動解除させることからなるステップを含む。

【００１０】本発明のもう一つの目的は、ジョブの実行を処理する装置を提案することにある。

【００１１】この目的を達成するために、オープン情報処理システム上でジョブの実行を処理する装置は、仮想メモリ、実メモリ、一時的なファイルスペース、最後の時間間隔の間のＣＰＵ使用時間として利用可能な資源を決定する手段と、予め他の要求に割り当てられている未使用の資源の量を算出する手段と、要求が提示されたジョブの実行に必要な資源の量と、予め他の要求に割り当てられた資源の量が差し引かれた利用可能な現在の資源の量とを比較する手段と、比較手段によって供給される結果に応じて、必要なジョブのスタート、延期またはスタートの拒否を制御する手段とを含むことを特徴とする。

【００１２】他の特徴によれば、ジョブの実行に必要な資源の量を特定のコマンドによって決定する手段を含む。

【００１３】別の特徴によれば、利用可能なシステム資源が充分にあるか否かを決定する機構を、作動させるあるいは作動解除させる手段を含む。

【００１４】本発明の他の特徴ならびに利点は、添付図に関してなされる以下の説明を読めば、より明らかになろう。

【００１５】

【発明の実施の形態】本発明のＵＮＩＸ上での実施態様を説明する前に、以下の定義を示しておくことが有益であろう。

【００１６】以下の説明において、「プロセス」または「ジョブ」という表現は、所与の瞬間におけるプログラムのあらゆる実行（従って、特にシステムにおけるその環境）を表し、プログラムは、それ自体が実行可能な通常のファイルとしてディスクに格納された不活性オブジェクトを構成する。「ユニックス」システムでは、次の二つのタイプのプロセスの存在が知られている。

【００１７】どの端末にも属さず、システムのスタートまたはシステムのマネージャによって決定される日付で作成され、システムを停止した場合にのみ中断されるシステムプロセス。たとえば、「スワッパ」、「デーモン（ｄａｅｍｏｎｓ）」として知られるいくつかのプロセスがある。これらは、「スプール」でのプリンタの適切な使用を保証するプロセス、あるいは所定の日にタスクをスタートさせることを可能にする「クロン」（ＣＲＯＮ）のような、プロセスである。

【００１８】特定のユーザにより所与の日付で所与の端末からスタートされるプロセス。特に、所定の識別データとともに端末に収容されることにより、各ユーザについて予め決定されたファイルの実行に対応するプロセスをスタートさせる。この種のプロセスは、殆ど常に制御言語のインタープリタ（ブルヌシェルまたはＣシェル）の実行に対応する。参考として述べると、ユーザによる標準サービスの利用は、遠距離にある機器により行われる。遠距離にある機器で機能するためには、「デーモン」と称される特定プロセスの存在を必要とするコマンドを介して行われる。

【００１９】図１に示された「ユニックス」システムの構造は、情報処理システム１のハードウェア資源からなり、この情報システムは、メモリおよび低レベルの入出力および様々なタスク（ジョブ）の連接を管理する「ユニックス」カーネル（ＫＥＲＮＥＬ）２と通信する。このカーネルの周囲で、「シェル」型制御言語の一つまたは複数の「インタープリタ」、電子メールシステム、および各種のユーティリティープログラムが用いられ、ユーティリティープログラムは勿論、Ｃ言語のコンパイラを含む。

【００２０】「ユニックス」システムにおけるこれらの従来の要素の周囲には、オープン環境にマルチタスクを適用する必要性から、２個の「ａｐｉ」プログラムモジュールが展開される。その一方は、ユーザによってスタートされたジョブの実行レポートを提供する（ＪＳＲ）であり、他方は、適切な経過あるいはトラブル発生をディスプレイし、従って原因を検出することの可能な（ＬＲＭ）である。最初のモジュールは、ジョブリポートサービス（Ｊｏｂ　ｒｅｐｏｒｔｉｎｇ　Ｓｅｒｖｉｃｅ）と称され、後のモジュールは、ローカル資源管理モジュール「ＬＲＭ」（Ｌｏｃａｌ　Ｒｅｓｓｏｕｒｃｅ　Ｍａｎａｇｅｍｅｎｔ」と称される。

【００２１】さらに本発明では、カーネル２およびアプリケーションプログラムのインターフェース３４と通信するローカル資源管理のデーモン３が付加される。このデーモン３は、ハードディスク内に記憶されるＬＲＭコンフィギュレーションファイル３３および関連するグラフィカルユーザーインターフェース（ＬＲＭ　ｇｕｉ）３２とも通信する。アプリケーションプログラムのアプリケーションインターフェース３４は、デーモン３および共有メモリ３１と通信する。共有メモリ３１はまた、グラフィカルユーザーインターフェース３２およびデーモン３と通信する。事前割り当ての必要性から、システ

5

ムは、ジョブの実行を報告するアプリケーションプログラム「ＪＲＳ」のインターフェース４４を組み込んでいる。このアプリケーション４４は、ハードディスクの特別なファイル５にジョブリポートを記憶するカタログ「ｊｏｒ」と通信する。アプリケーションプログラムのインターフェース４４は、プログラムが作動しており、ジョブリポートファイルが格納されていれば、ローカル資源管理デーモン３と通信し、資源の事前割り当ておよび動的調整を可能にする。インターフェース４４は、このインターフェースによりユーザが利用可能なコマンドを実行可能にするために必要なコードの行を含む。インターフェース４４により利用されるコマンドは次の通りである。

【００２２】ー資源のマネージャにコマンドを送り、リポートファイルの実行を発生することの可能な「ＳＪＲ」コマンド。アペンディックス１に示すこのコマンドは、仮想メモリのページ数、一時的なファイルのスペース量、ユーザによって決定されるディメンション、予想される、ＣＰＵの消費時間、最大経過時間、最大ＣＰＵ時間、一時的ファイルの最大スペース、仮想メモリページの最大数を決定し得る複数のオプションを含む。このコマンドは、上記の各種パラメータの意味と共に発明の詳細な説明のアペンディックス１に示す。

【００２３】「ＳＪＲｊｏｂｓｔａｒｔ」コマンド（ｉｎｔ　ｊｏｂｉｄ；　ｉｎｔ　ｑｆｌａｇ；　ｓｔｒｕｃｔ　ＳＪＲ＿ｒｅｓｏｕｒｃｅｓ　*ＳＪＲ＿ｒｅｓｏｕｒｃｅｓ；）、ここでｊｏｂｉｄパラメータはジョブの識別子であり、ＳＪＲ＿ｒｅｓｏｕｒｃｅｓはジョブに必要とされる資源を記述する構造体である。ジョブが資源の事前割り当てを必要としない場合、これは０となる。「ｑｆｌａｇ」パラメータは、それがないときには、ジョブが資源が使用可能になることを待つことを意味する。このコマンドは、ジョブがスタートできるかどうか検証し、スタートできない場合には、資源が利用可能になるまで待つが、これは「ｑｆｌａｇ」が０のときである。「ｑｆｌａｇ」が０でない場合は、直ちにリターンとなる。ジョブがスタート可能である場合は値０が返され、そうでなければ「ｑｆｌａｇ」が設定され、値－１が返される。さらに、特定のジョブの主要な各事象がカタログ「ｊｏｒ」のグローバルファイル「ｌｏｇｃ」に発生すると、これらの事象は記憶される。このファイルは、後述するパラメータｐ，ｔ，ｄ，ｃ，ｅ，Ｃ，Ｐ，Ｔの表示を可能にする「ｓｊｒ　ｌｏｇ　ｆｉｌｅ」コマンドによってディスプレイすることができる。ジョブの各エントリに対してパラメータを明示的に定義することを不要にするために、「ＳＪＲ」アプリケーション４４は、スクリプト内で資源宣言をする能力を提供する。かくしてアプリケーションの開発者が開発者のジョブに必要とされる資源の必要性を適切に規定した場合、これをスクリプトに挿入することができる。ジョブは、他のパラメータも持たない「ＳＪＲ」コマンドで入力される。このＳＪＲコマンドの構文は、以下の通りである。

6

【００２４】

＃ｏｐｔｉｏｎ－ｐ「仮想メモリページ数」
＃ｏｐｔｉｏｎ－ｔ「一時的ファイルのスペース量」
＃ｏｐｔｉｏｎ－ｄ「ユーザによって定義される機能」
＃ｏｐｔｉｏｎ－ｃ「予想されるＣＰＵ時間」
＃ｏｐｔｉｏｎ－ｅ「最大経過時間」
＃ｏｐｔｉｏｎ－Ｃ「ＣＰＵによる消費時間の限度」
＃ｏｐｔｉｏｎ－Ｐ「仮想メモリの最大ページの数」
＃ｏｐｔｉｏｎ－Ｔ「一時的ファイルの最大スペース」

衝突がある場合には、コマンドのエントリで特定された値は、次のように用いられる。スクリプトの内部に次のような一行があり、

＃ｏｐｔｉｏｎ－Ｃ２００

スクリプトが次のコマンドラインと共に入力される場合、

ＳＪＲ－Ｃ　３００　ｓｃｒｉｐｔ、

ＣＰＵの限度として用いられる値は３００である。

【００２５】このようにローカル資源管理モジュールを備えたシステムにおいて、重要でないプロセスは、重要なプロセスの実行の継続をできるだけ変えないようにしなければならない。各種のプロセスは、その固有の重要性を持っている。こうした重要性に応じて、またシステムの負荷に応じて多かれ少なかれプロセスに資源を割り当てなければならない。このような理由から「ＬＲＭ」モジュールは、プロセスをディメンションでソートすることができる。一つのディメンションは、ローカル資源管理の観点から同じ重要性を有する現在実行されるひと組のプロセスである。デフォルトで五個のディメンションが設けられている。即ち、第一のディメンション「システム（ＳＹＳＴＥＭ）」、第二のディメンション「ロット（ＬＯＴ）」（バッチ）、第三のディメンション「ダイバース（ＤＩＶＥＲＳ）」（ＭＩＳＣ）、第四のデータベースディメンション（ＤＢ）、および第五のトランザクション処理ディメンション（ＴＰ）である。通常のユーザによって直接バックグランドでプロセスがスタートされる場合、プロセスはディメンション「ダイバース」に属する。明示的なディメンションの宣言もなくタスク管理アプリケーションに入力されたスクリプトを介してプロセスがスタートされると、プロセスは、その場合、第二のディメンション「ロット」（バッチ）に属する。プロセスが他のいかなるディメンションにも属さない場合、プロセスは、「システム」ディメンションに属する。ディメンション「ＤＢ」および「ＴＰ」は、プロセスの定義がなされずに供給され、ユーザにより定義され得る。「システム」ディメンションを除いて、各ディメンションは、相対的な重み付けを有することが可能であり、この重み付けはファイルに記憶される。相対的

な重み付けは、ディメンションを含むプロセスの優先順位を管理するために用いる。一つのディメンションに属するあらゆるプロセスの優先順位は、同じように変化する。「システム」ディメンションは、相対的な重み付けを持たず、このディメンションに属するプロセスの優先順位は、ローカル資源管理（ＬＲＭ）によって変えられない。ディメンションによって制御されるプロセスは、常に、たとえシステムの負荷が非常に大きくても、それらの相対的な重み付けに応じて与えられる資源を有する。ローカル資源管理を開始する場合、あるいは、相対的な重み付けを特定することなく一つのディメンションが生成される場合、このディメンションは、デフォルトで－１の重み付けを有する。すなわち、このディメンションは存在するが、制御されないことを意味する。一つのディメンションは活動化されることもできれば非活動化されることもできる。ディメンションが非活動化されると、このディメンションに属する入力されたジョブはスタートされないが、このディメンションで実行中であるジョブは、実行され続ける。

【００２６】プロセスが、「ＳＪＲ」タスクの実行コマンドにより生じる場合、ディメンションが「ＳＪＲ」コマンド内で特定されれば、このプロセスは特定されたディメンションに属する。コマンド名およびユーザが一つのディメンションに関連する先祖（ａｎｃｅｓｔｏｒ）をプロセスが有する場合、プロセスはこのディメンションに属する。プロセスの実行時間の間、相対的な重み付けは、ディメンションを含むプロセスの優先順位を管理かつ変更するために用いられる。事前に割り当てられたプロセスは、ｓｊｒによってスタートされず、あるいは事前割り当てパラメータを含まない。

【００２７】資源管理アプリケーションによって、ユーザならびに資源マネージャは、各資源について、現在どのくらいの資源が使用されており、どのくらいの資源が各ジョブに宣言されているか、所与の時間に知ることができる。かくして、図２に示された例では、U^nは、事前割り当てのないプロセスによって使用される資源を示す。Ｒａは、管理プロセスおよびシステムに対して指定されている自由資源の量を示す。Ｕ１，Ｕ２は、ジョブ１および２によって使用される資源である。Ｄ１およびＤ２は、ジョブ１および２に宣言される資源を示す。Ｒ１は、Ｄ１とＵ１との差を示す。Ｒ２は、Ｄ２とＵ２との差を示し、Ａｆは、資源マネージャが利用可能なものとみなす資源を示す。ジョブが入力され、必要な資源の量Ｄ３を宣言すると、ＡｆがＤ３以上であればジョブスタートする。これは、必要最小限の資源が利用可能な場合にのみジョブのスタートを許容するという、資源の事前割り当ての有効性を示している。

【００２８】仮想メモリおよび一時的なファイルスペースの事前割り当てメカニズムは、システムの適切な機能に貢献するプロセスも構成する「ＳＪＲ」コマンドによって実行される。ジョブは、入力されると、ＳＪＲコマンドによって一定の資源の量を要求することができる。すると資源マネージャは、このジョブ用の資源が十分あるかどうか確かめる。ジョブは、最大経過時間、予想されるジョブの総計ＣＰＵ単位時間、と共に入力してもよい。資源の要求が調べられると、次の値が考慮される。

【００２９】要求される資源の量「ｒ」

現在、利用可能な資源の量「Ａ」

他の要求に既に事前割り当て済みであるが、未使用の資源「Ｒ」

制御されない資源割り当てによる故障を防止するためにユーザが資源マネージャに対しスタート時点で宣言した初期の資源の量「Ｒａ」

現在の資源の量「Ａ」は、定期的に得られる情報であり、記憶されたシステム情報テーブルを検証することによってＬＲＭ「デーモン」により定期的に更新される。既に事前割り当て済みである資源の値「Ｒ」もまた、ジョブによる現在の資源利用を取りまとめるＬＲＭ「デーモン」により決定される。「Ｒａ」は、これらの引数の一つとしてＬＲＭ「デーモン」によって供給される固定値である。

【００３０】ｒ＜Ａ－（Ｒ＋Ｒａ）である場合、要求は満たされる。各要求に対して、デーモンはこの計算を行い、比較の結果に応じて要求に応じたり、応じなかったりする。ジョブのスタートに対応する資源の要求が満たされると、ジョブは、その優先順位の変化を考慮に入れつつ実行される。

【００３１】下記のアペンディックスは、全体の実メモリおよび利用可能な実メモリを計算することを可能にするコード部分に関する。この計算は、下記のコード部分によって実行される。

【００３２】

【数１】

```
...Computing Available and Total Virtual and Real Memory.
     /*get_vmdata: Obtainment of Information on Available Real
and Virtual Memory */
get_vmdata(vm)
     struct vmdata *vm
{
     char      buff[1024];
     int     bufsiz = 1024;
     char      w1[20], w2[20], w3[20], w4[20];
     while (1){
     int       v1, v2;
     if(!init_access_to_commands_done){
/*Initialization: Command vmstat is launched, and its output
caught*/
          init_access_to_commands_done = 1;
          init_access_to_commands();
          fdvmcom = sag_popen_getline("vmstat 1", buff, &bufsiz,
2);
     } else {
          int cr;
/*A new line issued by command vmstat is read*/
          cr = sag_popen_nextline(fdvmcom, buff, *bufsiz, 2);
          if (cr <= 0){
               fdvmcom = sag_popen_getline("vmstat 1", buff,
*bufsiz, 2);
          }
     }
/*line is examined and information recovered*/
     sscanf(buff, "%s %s %s %s", w1, w2, w3, w4);
     if (!strcmp(w1, "kthr") && !strcmp(w2, "memory"))
          continue;
     if (!strcmp(w1, "-----") && !strcmp(w2, "----------"))
          continue;
     if (!strcmp(w1, "r") && !strcmp(w2, "b"))
          continue;
     sscanf(w3, "%d", &v1);
     sscanf(w4, "%d", &v2);
     vm->totrealmem = vmconstant.totrealmem;
/*System Call psdanger returns Total Virtual Memory and Available
Virtual Memory*/
     vm->totvirtmem = psdanger(0) * 4;
     vm->avrealmem = v2 * 4;
     vm->avvirtmem = psdanger(-1) * 4;
     return (0);
  }
}
```

【００３３】プロセスはまた、利用可能な全体の一時的ファイルスペースの計算ステップを含む。計算ステップは、以下のコード部分によって実行される。

＊【００３４】【数２】＊

```
Computing Available and Total Temporary File Space

updtmpspace(avtmpspace, tottmpspace)
     int       *avtmpspace;
     int       *tottmpspace;
{
     int       curnbtmpdev;
     char      *ptdevlist;
     int       i;
     struct statfs  statfsbuf;

     curnbtmpdev = enum_dev();
     if (curnbtmpdev > nbtmpdev){
          nbtmpdev = curnbtmpdev;
          if (tmpdevlist)
               (unsigned char *) free(tmpdevlist);
          tmpdevlist = (char*) malloc(nbtmpdev *200);
          tmpdevno = (int *) malloc(nbtmpdev *sizeof(int));
     }
     if (curnbtmpdev > 0){
          get_dev(tmpdevlist, tmpdevno);
          *avtmpspace = 0;
          *tottmpspace = 0;
          for (i = 0, ptdevlist = tmpdevlist; i < curnbtmpdev;
i++, ptdevlist += 200){
               int    cr;
/*System Call ststfs provides information on the File System*/
               cr = statfs(ptdevlist, &statfsbuf);
               /*conversion into Kb: multiply by 4*/
               *avtmpspace += 4 * statfsbuf.f_bfree;
               *tottmpspace += 4 * statfsbuf.f_blocks;
          }
     }
}
```

【００３５】次に、最後時間間隔の間に生じたＣＰＵの利用を決定するが、この決定は、以下のコード部分を用いて行われる。

【００３６】【数３】

11 12

```
        /*sysinfo and vminfo are AIX Kernel tables*/
        /*The function get_sysinfo_and_vminfo reads these
tables from the Kernel Memory*/
        get_sysinfo_andvminfo (&sysinfo, &vminfo);
        tpgspgouts      =     (double)  (vminfo.pgspgouts   -
        ovminfo.pgspgouts)/(double) Irmdinterval;
        trunque   =  (double) (sysinfo.runque -
        oysinfo.runque)/(double) Irmdinterval;
        tswpque   =  (double) (sysinfo.swpque-
        osysinfo.swpque)/(double) Irminterval;

        tuser = (double) (sysinfo.cpu[CPU_USER])/(double)
        Irmdinterval;
        tsystem    =     (double)  (sysinfo. cpu[CPU_KERNEL]-
        osysinfo.cpu[CPU_KERNEL]/(double) Irmdinterval;
        tidle   =  (double)  (sysinfo.cpu[CPU_IDLE]-
        osysinfo.cpu[CPU_IDLE]-/(double)Irmdinterval;
        twait      =     (double)       (sysinfo.cpu[CPU_WAIT]-.
        osysinfo.cpu[CPU_WAIT]/(double) Irmdinterval;
        tsum = tuser + tsystem + tidle + twait;
```

【００３７】プロセスはまた、事前割り当てされている　＊から実行される。
が、タスクによってまだ使用されていないスペースの決　【００３８】
定ステップを含む。このステップは、以下のコード部分＊20　【数４】

```
        Computing Prealloc space not used by jobs:

        shmem->system.preallocvirtmem = 0;
        shmem->system.prealloctmpspace = 0;
        for (job = shmem->first_jb_started; job; job = job->next){
             if (job->requiredvirtmem && (job->requiredvirtmem >
job->currentvirtmem)){
                  shmem->system.preallocvirtmem += (job-
>requiredvirtmem - job->current virtmem);
             }
             if (job->requiredtmpspace && (job->requiredtmpspace >
job->currentusedtmpspace))
                  shmem->system.prealloctmpspace += (job-
>requiredtmpspace - job->currentusedtmpspace);
        }
```

【００３９】評価は、式Ｒ＜（Ｒ＋Ｒａ）によって実行　30　【００４０】
され、以下のコード部分からなる。　【数５】

13 14

```
LRMresourcescheck (LRMresources)

     struct LRMresources *LRMresources;
{
     struct timeval tv;
     struct timezone tz;
     struct systementry system;

     gettimeofday(&tv, &tz);
     shmgetsystem(&system);
     if (LRMresources->requiredtmpspace > (system.avtmpspace -
system.prealloctmpspace - system.reservedtmpspace))                    35
          return (0);
     if (LRMresources ->requiredvirtmem > (system.avvirtmem -
system.preallocvirtmem - system.reservedvirtmem))                      36
          return (0);
     if (LRMresources->expectedtermination){
          int       remains;
          int       tl;
          double         a, b;
          remains-LRMresources->expectedtermination - tv.tv_sec;
          LRMresources->neededratecpu = 100.0 * LRMresources->
expectedcpu / remains;                                                 37
          /*
           * test here with the current available cpu rate
           */
          LRMdbgprintf(DBGALL, "Maximum Elapsed time specified\n");
          LRMdbgprintf(DBGALL, "Remains: %d\n", remains);
          LRMdbgprintf(DBGALL, "Needed Rate of CPU: %f\n",
LRMresources->neededratecpu);
          LRMdbgprintf(DBGALL, "Extrapolated Available Rate
of CPU: %d\n", system.workingavcpu);
          if (remains > 0 && LRMresources->neededratecpu <=
system.workingavcpu){
            LRMdbgprintf(DBGALL, "Rate of CPU is good enough\n");
            return (1);
          } else {
            LRMdbgprintf(DBGALL, "Rate of CPU is too low\n");
            if (remains < 0 | LRMresources->neededratecpu >
100.0* system.nbprocessors){
              LRMdbgprintf(DBGALL, "There will not be enough
free CPU to complete the job in time\n");
              return (-1);                                             38
            }
            return (0);
          }
     } else
          return (1);
}
```

【００４１】このコード部分では、部分３５が一時的なスペース部分の評価を行い、充分な一時的スペースが利用可能である場合は、値０を返す。次に、プロセスは、部分３６によって実行される仮想メモリ部分について評価を続行し、充分な仮想メモリスペースが利用可能である場合は、値０を返す。最後にシステムは、コード部分３８が利用できるＣＰＵ資源の評価を行う。この評価は、残りの資源（ＲＥＭＡＩＮＳ）と必要とされるＣＰＵ率（ＮＥＥＤＥＤ　ＲＡＴＥ　ＣＰＵ）とを決定した後で行われる。利用可能なＣＰＵ率が充分である場合、システムは値１に戻るので要求が実行可能になるが、システムが値－１に戻る場合は、ジョブの実行の要求が延期される。

【００４２】このようにして、ジョブの実行に必要なパラメータの定義をユーザが予めＳＪＲコマンドに入力しておくことにより、ユーザは、上記の機構を利用して、ジョブが、その適切な実行に必要な資源を有するとき、システムでジョブの実行をスタートすることができる。

【００４３】当業者が考慮しうるあらゆる変更も同様に、本発明の一部をなす。従って、上記の方法および装置は、たとえば「ウィンドウズＮＴ」などの他のオープンシステムに容易に使用かつ適用可能である。

【００４４】アンペディックス１

ＮＡＭＥ：ｓｊｒコマンド

ｓｊｒ－資源管理にコマンドを入力し、ＳＹＮＯＰＴＩＱＵＥファイルに実行リポートを発生する。

【００４５】ｓｊｒ［－ｐ＜仮想メモリページ数］

［－ｔ＜一時的なファイルスペース数］

［－ｑ］

［－ｆ］

［－ｌ］

［－ｎ］

［－ｏ］

［－ｄ＜ユーザにより定義されるディメンション＞］

［－ｃ＜予想されるＣＰＵ消費時間＞－ｅ＜最大経過時間］

［－Ｃ＜最大ＣＰＵ時間＞］

［－Ｔ＜一時的なファイルの最大スペース＞］

［－Ｐ＜最大仮想メモリページ数＞］

［－ｒ０｜１｜２｜３］

［－ｖ＜ｅｎｖｉｒｏｎｍｅｎｔｖａｒｉａｂｌｅ＞＝＜ｖａｌｕｅ＞］

［ｃｏｍｍａｎｄ　ａｒｇ・・・］

【図面の簡単な説明】

【図１】情報処理システムおよび、本発明の方法を実施可能な情報処理システムに関連づけられるソフトウェア手段を示す概略図である。

【図２】本発明による資源管理における状態の一例を示す図である。

【符号の説明】

15

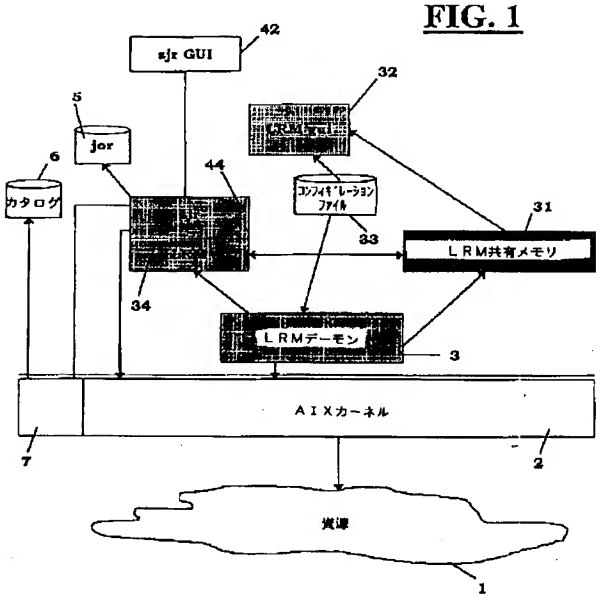
1　資源
2　カーネル「ユニックス」
3　ＬＲＭ（ローカル資源管理）デーモン
5　ｊｏｒファイル
6　カタログ
３１　ＬＲＭ共有メモリ

16

＊３２　ＬＲＭ　ｇｕｉ
３３　ＬＲＭコンフィギュレーションファイル
３４　ＬＲＭアプリケーションインターフェース
４２　ｓｊｒ　ＧＵＩ
４４　ＳＪＲアプリケーションインターフェース
＊

【図１】

FIG. 1



【図２】

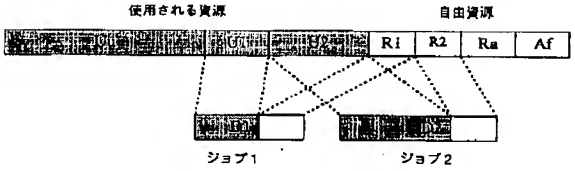


FIG. 2